Troisième partie

Rendre la recherche réutilisable et interopérable : *KatAPI*, une API pour échanger des données structurées

La problématique de cette partie est la suivante : comment rendre la recherche en humanités numériques réutilisable et encourager le partage de données entre projets de recherche ? La réponse à cette question, au sein du projet *MSS / Katabase* prend la forme d'une API. Celle-ci diffuse des données source du projet, mais aussi des informations issues de la recherche et du traitement computationnel de celles-ci dans deux formats structurés : le `JSON` et le `XML-TEI`.

Chapitre 8

Standards de design et statut des API dans pour les humanités numériques centrées sur le texte

Ici est présenté un état de l'art sur la conception d'API dans un contexte d'humanités numériques. Sont présentés un standard de design pour la conception d'api (le REST), l'architecture API-DTS (équivalent du IIIF pour le texte). Si l'API conçue ne se rattache pas à ces concepts, ceux-ci ont servi d'inspiration pour l'architecture de *KatAPI*. Les principes FAIR (essentiels pour la recherche ouverte et interopérable) sont également présentés, ainsi que l'état de l'art pour le design d'API dans un contexte d'humanités numériques. Il apparaît que de tels outils sont majoritairement conçus par des institutions (*DataBnF*...) plutôt que par de simples projets de recherche.

8.1 Pourquoi partager les données de la recherche ? Les principes FAIR

Les principes *Findable Accessible Interoperable Reusable* (FAIR) [1], qui visent à établir un ensemble de caractéristiques que les données ouvertes doivent partager, sont présentés ici.

1. Martin Boeckhout, Gerhard A. Zielhuis et Annelien L. Bredenoord, « The FAIR guiding principles for data stewardship : fair enough ? », *European Journal of Human Genetics*, 26–7 (juil. 2018), p. 931-936, DOI : 10.1038/s41431-018-0160-0.

8.2 Le standard REST : un modèle pour la conception d'API

Ici est présenté le standard REST, défini par Roy Fielding [2] ; ce standard architectural est considéré comme l'idéal à atteindre en matière de design d'API. Il n'a cependant pas été suivi ici, en partie parce qu'il est mieux adapté à un projet de plus grande échelle, et en partie car il ne fonctionne pas de façon optimale avec le modèle de données défini, et notamment avec le choix de construire une API renvoyant des réponses conformes à la `TEI`.

8.3 CTS, OAI-PMH et DTS : quels standards pour le partage du texte en humanités numériques ?

Les principes REST, présentés ci-dessus, concernent l'architecture et le design d'une API ; ils ne définissent pas précisément quels formats de réponse définir, ni comment structurer ses (méta-)données. Ici sont présentés quelques protocoles visant à uniformiser les formats de réponse et l'interaction avec des API. Ils visent à encourager la standardisation et l'interopérabilité. Trois standards sont présentés en détail : le *Canonical Text Services* (CTS) [3], *Open Archives Initiative Protocol for Metadata Harversting* (OAI-PMH) [4] et *Distributed Text Services* (DTS) [5]. Comme le REST, ces standards sont difficiles à intégrer à un petit projet de recherche ; ils peuvent cependant servir d'inspiration pour la conception d'une API centrée sur le texte.

2. R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*…

3. Bridget Almas et Thibault Clérice, « Continuous Integration and Unit Testing of Digital Editions », *Digital Humanities Quarterly*, 11–4 (févr. 2018), Publisher : Alliance of Digital Humanities, URL : `https://hal.archives-ouvertes.fr/hal-01709868`.

4. Camille Prime-Claverie et Annaïg Mahé, « Le défi de l'interopérabilité entre plates-formes pour la construction de savoirs augmentés en sciences humaines et sociales », dans *Ecrilecture augmentée dans les communautés scientifiques*, 2017, URL : `https://archivesic.ccsd.cnrs.fr/sic_01511618`.

5. B. Almas, T. Clérice, Hugh Cayless, V. Jolivet, Pietro Maria Liuzzo, Matteo Romanello, Jonathan Robie et Ian W. Scott, *Distributed Text Services (DTS) : a Community-built API to Publish and Consume Text Collections as Linked Data*, mars 2021, URL : `https://hal.archives-ouvertes.fr/hal-03183886`.

Chapitre 9

Définir un périmètre : que partager, et comment partager ?

Après avoir présenté le paysage des API en humanités numériques, ce chapitre décrit *KatAPI* du point de vue de l'utilisateur.ice : quelles données peuvent être obtenues via l'API, quels paramètres de recherche sont autorisés et quels sont les formats de données retournés par l'application. C'est donc les principes de la communication du client avec l'API qui sont ici présentés.

9.1 Quelles données partager ?

Ici est défini le périmètre des données accessibles depuis l'API : tout ce qui est produit par le projet n'est bien sûr pas accessible, et des choix ont dû être faits. Les jeux de données accessibles sont présentés ici.

9.2 Codifier l'accès aux données : présentation des paramètres de recherche possibles

Ici sont présentés les paramètres de recherche qui peuvent être utilisés pour récupérer des données, et comment ils permettent de filtrer ce qui est retourné par l'API.

9.3 Comment partager les données ? Principes suivis pour le partage d'informations, formats et structure des réponses de l'API

Cette section décrit les formats de réponse de l'API, en `JSON` et `XML-TEI`, pour les différents paramètres de requête définis.

Chapitre 10

Implémentation et fonctionnement technique de *KatAPI*

Ce chapitre décrit le fonctionnement de l'API côté serveur : comment les données sont reçues, la manière dont elles sont traitées et comment les réponses sont construites (notamment la création automatisée de documents `XML-TEI`). Le système de gestion des erreurs est également présenté.

10.1 Présentation générale

Ici est présenté, schéma à l'appui, le fonctionnement technique de l'API et la chaîne de traitement depuis la réception des requêtes jusqu'à l'envoi d'une réponse.

10.2 Gestion des erreurs

L'API étant une application devant interagir avec des utilisateur.ice.s et gérer leurs requêtes, il n'est pas pensable qu'elle « plante » ou ne renvoie pas d'erreur. Tout un système de gestion des erreurs a donc été défini. Ces erreurs peuvent être classées en deux catégoriés : côté client (dues à des entrées invalides des utilisateur.ice.s) et côté serveur (erreurs inattendues qui empêchent l'exécution d'une requête). Si ces erreurs sont rencontrées, des réponses sont construites en `JSON` ou `XML-TEI` (selon le format demandé par l'utilisateur.ice.s). Celles-ci décrivent l'erreur et permettent donc aux utilisateur.ice.s de corriger les erreurs qui ont pu avoir lieu côté client.

10.3 Garantir le bon fonctionnement de l'application

De la même manière que les systèmes de gestion des erreurs garantissent que l'application continue à fonctionner même en cas d'erreur, les tests garantissent que l'API soit

fonctionnelle, même dans « des conditions extrêmes ». Les réponses, formats et données retournées sont donc testés. C'est ce protocole de test qui est présenté ici.